3. Auflage

Brendan Burns · Joe Beda
Kelsey Hightower · Lachlan Evenson

Kubernetes

Eine kompakte Einführung

dpunkt.verlag

Brendan Burns begann seine Karriere mit einem kurzen Einsatz in der Software-Branche, bevor er sich mit einem PhD in Robotik auf die Bewegungsplanung für menschenähnliche Roboterarme konzentrierte. Darauf folgte eine kurze Zeit als Informatik-Professor. Schließlich kehrte er nach Seattle zurück und kam zu Google, wo er an der Web-Suchinfrastruktur mit einem Schwerpunkt auf Low-Latency Indexing arbeitete. Dort gründete er auch zusammen mit Joe Beda und Craig McLuckie das Kubernetes-Projekt. Brendan Burns ist aktuell Director of Engineering bei Microsoft Azure.

Joe Beda begann seine Karriere bei Microsoft am Internet Explorer (er war jung und naiv). Während der sieben Jahre bei Microsoft und der zehn Jahre bei Google hat Joe Beda an GUI-Frameworks, Echtzeit-Sprache und Chat, Telefonie, maschinellem Lernen für Anzeigen und Cloud Computing gearbeitet. Vor allem aber hat er bei Google die Google Compute Engine aus der Taufe gehoben und zusammen mit Brendan Burns und Craig McLuckie Kubernetes geschaffen. Zusammen mit McLuckie gründete Beda das Start-up Heptio, das sie an VMware verkauften und bei dem er nun Principal Engineer ist. Auf Seattle als seine Heimat ist er sehr stolz.

Kelsey Hightower ist Principal Developer Advocate bei Google, wo er an deren Cloud-Plattform arbeitet. Er half bei der Entwicklung und dem Verbessern vieler Google-Cloud-Produkte – unter anderem bei Googles Kubernetes Engine, Cloud Functions und dem API Gateway von Apigees. Hightower verbrachte einen Großteil seiner Zeit mit Executives und Entwicklern aus vielen Fortune-1000-Unternehmen und half ihnen dabei, mithilfe der Technologien und Plattformen von Google ihre eigenen Geschäfte voranzubringen. Er trägt viel zu Open-Source-Projekten bei und betreut Projekte, die Software-Entwickler und Operations-Professionals beim Aufbauen und Ausliefern von Cloud-Native-Anwendungen helfen. Hightower ist bekannter Autor und Keynote-Sprecher und war der erste Gewinner des CNCF Top Ambassador Awards aufgrund seiner Unterstützung beim Aufsetzen der Kubernetes-Community. Er ist Mentor und Technical Advisor und hilft Gründern dabei, ihre Visionen Realität werden zu lassen.

Lachlan Evenson ist Principal Program Manager für das Open-Source-Team bei Azure. Er ist aktives Mitglied der Kubernetes-Community und hat im Steering Committee als Release Lead agiert. Lachlan Evenson besitzt ein umfassendes operationales Wissen für viele Cloud-native Projekte und er verbringt seine Tage damit, im Cloud-nativen Ökosystem an Open-Source-Projekten zu bauen und zu ihnen beizutragen.

Brendan Burns · Joe Beda · Kelsey Hightower · Lachlan Evenson

Kubernetes

Eine kompakte Einführung

3., aktualisierte und erweiterte Auflage

dpunkt.verlag

Brendan Burns
Joe Beda
Kelsey Hightower
Lachlan Evenson

Übersetzung: Thomas Demmig
Lektorat: Sandra Bollenbacher
Copy-Editing: Petra Heubach-Erdmann, Düsseldorf
Satz: inpunkt[w]o, Haiger, *www.inpunktwo.de*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Helmut Kraus, *www.exclam.de*
Druckerei: mediaprint solutions GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:
Print 978-3-86490-959-7
PDF 978-3-96910-962-5
ePub 978-3-96910-963-2
mobi 978-3-96910-964-9

3., aktualisierte und erweiterte Auflage 2023

dpunkt.verlag GmbH
Wieblinger Weg 17
69123 Heidelberg

Authorized German translation of the English edition of *Kubernetes: Up and Running, 3E*
ISBN 9781098110208 © 2022 Brendan Burns, Joe Beda, Kelsey Hightower, and Lachlan Evenson.
This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Hinweis:
Dieses Buch wurde mit mineralölfreien Farben auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie. Hergestellt in Deutschland.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: *hallo@dpunkt.de*.

5 4 3 2 1 0

Für Robin, Julia, Ethan und alle, die Cookies gekauft hatten, damit ich mir in der dritten Klasse einen Commodore 64 leisten konnte.

– Brendan Burns

Für meinen Vater, durch den ich Computer lieben lernte, indem er Lochkarten und Punktmatrix-Banner mit nach Hause brachte.

– Joe Beda

Für Klarissa und Kelis, durch die ich auf dem Teppich bleibe. Und für meine Mutter, die mich ein strenges Arbeitsethos gelehrt hat und mir zeigte, wie ich Widerstände überwinden kann.

– Kelsey Hightower

Für Mum und Dad, die mir ein starkes Selbstbewusstsein und viel Neugierde eingeflößt haben, die mich dazu bringt, alles zu lernen, wonach mir ist.

– Lachlan Evenson

Inhalt

Einleitung

Kubernetes möchte jedem Sysadmin danken, der um 3 Uhr in der Früh geweckt wurde, um einen Prozess neu zu starten. Jedem Entwickler, der Code in die Produktivumgebung geschoben hat, um dann festzustellen, dass er dort nicht wie auf dem eigenen Laptop lief. Jedem Systemarchitekten, der unabsichtlich einen Lasttest gegen den Produktivserver laufen ließ, weil irgendein Hostname nicht angepasst wurde. Dieser Schmerz, diese unfreundlichen Arbeitszeiten und diese verrückten Fehler haben die Entwicklung von Kubernetes inspiriert. Kurz: Kubernetes will das Bauen, Deployen und Warten verteilter Systeme radikal vereinfachen. Es wurde durch die jahrzehntelange Erfahrung beim Bauen zuverlässiger Systeme inspiriert und ist von Grund auf so entworfen, dass sein Einsatz vielleicht nicht euphorisch macht, aber zumindest erfreut. Wir hoffen, Sie haben an diesem Buch Spaß!

Wer dieses Buch lesen sollte

Ob Sie mit verteilten Systemen noch keine Erfahrung haben oder schon seit Jahren Cloud-native Systeme deployen – Container und Kubernetes können Ihnen dabei helfen, in Bezug auf Geschwindigkeit, Agilität, Zuverlässigkeit und Effizienz in ganz neue Bereiche vorzustoßen. Dieses Buch beschreibt den Cluster-Orchestrierer Kubernetes und die Anwendung seiner Tools und APIs, um die Entwicklung, Auslieferung, Sicherheit und Wartung verteilter Anwendungen zu verbessern. Es wird zwar keine Erfahrung mit Kubernetes vorausgesetzt, aber um den größtmöglichen Nutzen aus diesem Buch zu ziehen, sollten Sie mit dem Bauen und Deployen von serverbasierten Anwendungen vertraut sein. Wenn Sie Konzepte wie Load Balancer und Network Storage kennen, ist das nützlich, aber nicht zwingend erforderlich. Genauso ist Erfahrung mit Linux, Linux-Containern und Docker zwar nicht essenziell, aber sie hilft Ihnen, um das Buch möglichst gut einsetzen zu können.

Warum wir dieses Buch geschrieben haben

Wir haben mit Kubernetes seit seinen Anfängen zu tun. Es war wirklich erstaunlich, seine Entwicklung von einer Spielerei, die vor allem experimentell genutzt wurde, hin zu einer zentralen, produktionsreifen Infrastruktur zu beobachten, die produktive Anwendung im großen Maßstab in vielen Bereichen betreibt. Auf diesem Weg wurde immer deutlicher, dass ein Buch mit den zentralen Konzepten von Kubernetes und der Motivation hinter der Entwicklung dieser Konzepte für die aktuelle Cloud-native Anwendungsentwicklung ein wichtiger Beitrag wäre. Wir hoffen, dass Sie mit dem Lesen dieses Buches nicht nur lernen, wie Sie zuverlässige und skalierbare Anwendungen auf Basis von Kubernetes bauen, sondern auch Einblicke in die zentralen Herausforderungen verteilter Systeme erlangen, die zu dessen Entwicklung geführt haben.

Warum wir dieses Buch aktualisiert haben

In den Jahren seit dem Erscheinen der ersten und zweiten Auflage dieses Buches ist das Ökosystem von Kubernetes weiter gewachsen und hat sich fortentwickelt. Es gab viele Releases von Kubernetes, und viele zusätzliche Tools und Patterns für den Einsatz von Kubernetes wurden zu De-facto-Standards. In der dritten Auflage haben wir uns darauf konzentriert, Themen hinzuzunehmen, die im Ökosystem von Kubernetes wachsendes Interesse verzeichnen, wie zum Beispiel Sicherheit, den Zugriff über Programmiersprachen und das Deployen in mehrere Cluster. Auch haben wir die bestehenden Kapitel auf den neuesten Stand gebracht, um die Änderungen und die Weiterentwicklung von Kubernetes widerzuspiegeln. Wir gehen fest davon aus, dieses Buch in ein paar Jahren erneut überarbeiten zu müssen (und freuen uns darauf), wenn Kubernetes weiter seinen Weg geht.

Ein Wort zu aktuellen Cloud-nativen Anwendungen

Von den ersten Programmiersprachen über die objektorientierte Programmierung bis hin zur Entwicklung der Virtualisierung und Cloud-Infrastruktur ist die Geschichte der Informatik auch eine Geschichte der Entwicklung von Abstraktionen, die Komplexität verbergen und Sie in die Lage versetzen, immer ausgefeiltere Anwendungen zu bauen. Trotzdem ist das Entwickeln zuverlässiger, skalierbarer Anwendungen immer noch eine viel größere Herausforderung, als es sein sollte. In den letzten Jahren hat sich gezeigt, dass Container und zugehörige Orchestrierungs-APIs wie Kubernetes zu einer wichtigen Abstraktion wurden, die die Entwicklung zuverlässiger, skalierbarer und verteilter Systeme radikal vereinfacht hat. Container und Orchestrierer ermöglichen es Entwicklern, Anwendungen mit einer Schnelligkeit, Agilität und Zuverlässigkeit zu bauen, die vor ein paar Jahren noch als unerreichbare Zukunftsmusik gegolten hätten.

Was Sie in diesem Buch finden

Dieses Buch ist wie folgt organisiert. Kapitel 1 umreißt auf allgemeinem Niveau die Vorteile von Kubernetes, ohne allzu sehr in die Details zu gehen. Wenn Kubernetes für Sie neu ist, hilft Ihnen dieses Kapitel, zu verstehen, warum Sie den Rest des Buches lesen sollten.

Kapitel 2 liefert eine detaillierte Einführung in Container und die containerisierte Anwendungsentwicklung. Wenn Sie noch nie mit Docker gespielt haben, wird dieses Kapitel eine nützliche Einführung sein. Sind Sie schon ein Docker-Experte, wird es sich für Sie eher um ein Review handeln.

Kapitel 3 behandelt das Deployen von Kubernetes. Während sich ein Großteil dieses Buches darum dreht, wie Sie Kubernetes *einsetzen*, brauchen Sie ein lauffähiges Cluster, bevor Sie loslegen können. Das Betreiben eines Clusters für eine Produktiv-Umgebung liegt außerhalb des Rahmens dieses Buches, aber in diesem Kapitel stellen wir ein paar einfache Wege vor, ein Cluster so aufzusetzen, dass Sie verstehen können, wie Sie Kubernetes einsetzen. Kapitel 4 beschreibt eine Auswahl von gebräuchlichen Befehlen, die zur Interaktion mit Kubernetes-Clustern eingesetzt werden.

Ab Kapitel 5 kümmern wir uns um die Details des Deployens einer Anwendung mit Kubernetes. Wir beschreiben Pods (Kap. 5), Labels und Anmerkungen (Kap. 6), Services (Kap. 7), Ingress (Kap. 8) und ReplicaSets (Kap. 9). Diese bilden die Grundlagen für das Deployen Ihres Service in Kubernetes. Dann wenden wir uns Deployments zu (Kap. 10), die den Lebenszyklus einer kompletten Anwendung verbinden.

Im Anschluss daran geht es um speziellere Objekte in Kubernetes: DaemonSets (Kap. 11), Jobs (Kap. 12) sowie ConfigMaps und Secrets (Kap. 13). Diese Kapitel sind zwar für viele produktive Anwendungen wichtig, aber wenn Sie Kubernetes gerade erst kennenlernen, können Sie sie überspringen und sich später mit ihnen beschäftigen, wenn Sie mehr Erfahrung und Expertise erlangt haben.

In Kapitel 14 stellen wir dann die Role-Based Access Control (RBAC) vor und kümmern uns um Service Meshes (Kap. 15) und das Integrieren von Storage in Kubernetes (Kap. 16). Wir beschreiben das Erweitern von Kubernetes (Kap. 17) und den Zugriff über Programmiersprachen (Kap. 18). Dann geht es um das Absichern von Pods (Kap. 19) und Policy und Governance in Kubernetes (Kap. 20). Zum Schluss stellen wir noch Beispiele für das Entwickeln und Deployen von Multicluster-Anwendungen (Kap. 21) vor und wir beschreiben, wie Sie Ihre Anwendungen unter Versionskontrolle organisieren können (Kap. 22).

Online-Ressourcen

Sie sollten Docker (*https://docker.com*) installieren. Auch werden Sie sich mit der zugehörigen Dokumentation vertraut machen wollen, wenn Sie das noch nicht getan haben.

Genauso sollten Sie das Befehlszeilen-Tool `kubectl` (*https://kubernetes.io*) installieren und dem Slack-Channel Kubernetes beitreten (*https://slack.kubernetes.io*), wo Sie eine große Community vorfinden, die nahezu rund um die Uhr zum Reden und Beantworten von Fragen bereitsteht.

Wenn Sie schließlich mehr Erfahrung gesammelt haben, können Sie sich auch mit dem Open-Source-Repository von Kubernetes auf GitHub vertraut machen (*https://github.com/kubernetes/kubernetes*).

Konventionen in diesem Buch

Die folgenden typografischen Konventionen werden in diesem Buch genutzt:

- *Kursiv*
  Für neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateierweiterungen.
- `Nichtproportionalschrift`
  Für Programmlistings, aber auch für Codefragmente in Absätzen, wie etwa Variablen- oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter.
- **`fette Nichtproportionalschrift`**
  Für Befehle und anderen Text, der genau so vom Benutzer eingegeben werden sollte.
- ***`kursive Nichtproportionalschrift`***
  Für Text, der vom Benutzer durch eigene Werte ersetzt werden sollte.

Tipp

Dieses Symbol steht für einen Tipp, Vorschlag oder allgemeinen Hinweis.

Warnung

Dieses Symbol steht für eine Warnung oder Vorsichtsmaßnahme.

Der Einsatz von Codebeispielen

Die aktuellen Codebeispiele zu diesem Buch finden Sie zum Herunterladen auf *https://dpunkt.de/produkt/kubernetes-3*.

Dieses Buch ist dazu da, Ihnen beim Erledigen Ihrer Arbeit zu helfen. Im Allgemeinen dürfen Sie die Codebeispiele aus diesem Buch in Ihren eigenen Programmen und der dazugehörigen Dokumentation verwenden. Sie müssen uns dazu nicht um Erlaubnis fragen, solange Sie nicht einen beträchtlichen Teil des Codes reproduzieren. Beispielsweise benötigen Sie keine Erlaubnis, um ein Programm zu schreiben, in dem mehrere Codefragmente aus diesem Buch vorkommen. Wollen Sie dagegen einen Datenträger mit Beispielen aus Büchern von der dpunkt.verlag GmbH verkaufen oder verteilen, benötigen Sie eine Erlaubnis. Eine Frage zu beantworten, indem Sie aus diesem Buch zitieren und ein Codebeispiel wiedergeben, benötigt keine Erlaubnis. Eine beträchtliche Menge Beispielcode aus diesem Buch in die Dokumentation Ihres Produkts aufzunehmen, bedarf hingegen einer Erlaubnis.

Wir freuen uns über Zitate, verlangen diese aber nicht. Ein Zitat enthält Titel, Autor, Verlag und ISBN. Beispiel: »*Kubernetes* von Brendan Burns, Joe Beda, Kelsey Hightower und Lachlan Evenson. Copyright 2023 dpunkt.verlag GmbH, 978-3-86490-959-7.«

Wenn Sie glauben, dass Ihre Verwendung von Codebeispielen über die übliche Nutzung hinausgeht oder außerhalb der oben vorgestellten Nutzungsbedingungen liegt, kontaktieren Sie uns bitte unter *hallo@dpunkt.de*.

Wie Sie uns erreichen

Mit Anmerkungen, Fragen oder Verbesserungsvorschlägen zu diesem Buch können Sie sich jederzeit an den Verlag wenden:

hallo@dpunkt.de

Bitte beachten Sie, dass über unsere E-Mail-Adresse kein Software-Support angeboten wird.

Danksagungen

Wir möchten uns bei allen bedanken, die zum Entstehen dieses Buches beigetragen haben. Dazu gehören unsere Lektorinnen Virginia Wilson und Sarah Grey sowie all die tollen Leute bei O'Reilly, aber auch die technischen Korrektoren, die so viel Feedback geliefert und das Buch damit deutlich verbessert haben. Schließlich möchten wir uns noch bei allen Lesern der ersten und zweiten Auflage bedanken, die sich die Zeit genommen haben, Fehler einzusenden, die wir in der dritten Auflage beheben konnten. Vielen Dank an alle!

1 Einführung

Kubernetes ist ein Open-Source-Orchestrierer für das Deployen containerisierter Anwendungen. Es wurde ursprünglich von Google entwickelt und ist durch ein Jahrzehnt Erfahrung beim Deployen skalierbarer, zuverlässiger Systeme in Containern über anwendungsorientierte APIs inspiriert.[1]

Seit seiner Premiere im Jahr 2014 hat sich Kubernetes zu einem der weltweit größten und erfolgreichsten Open-Source-Projekte gemausert. Es wurde zur Standard-API für das Erstellen Cloud-nativer Anwendungen und ist für so gut wie jede öffentliche Cloud verfügbar. Kubernetes bietet eine gut getestete Infrastruktur für verteilte Systeme, die für Cloud-native Entwickler in allen Maßstäben passt – von einem Cluster aus Raspberry Pis bis hin zu einem Rechenzentrum voller leistungsfähiger, moderner Rechner. Es liefert die Software, die notwendig ist, um zuverlässige, skalierbare verteilte Systeme zu bauen und zu deployen.

Sie fragen sich vielleicht, was wir meinen, wenn es um »zuverlässige, skalierbare verteilte Systeme« geht. Mehr und mehr Services werden über das Netzwerk per API bereitgestellt. Diese APIs werden oft durch ein *verteiltes System* bedient, also den diversen Elementen, die die API implementieren und auf verschiedenen Rechnern laufen, die über das Netz verbunden sind und ihre Aktionen per Netzwerk-Kommunikation koordinieren. Weil wir uns in allen Aspekten unseres täglichen Lebens zunehmend auf diese APIs verlassen (zum Beispiel, den Weg zum nächsten Krankenhaus zu finden), müssen diese Systeme ausgesprochen *zuverlässig* sein. Sie dürfen keine Ausfälle haben, auch dann nicht, wenn ein Teil des Systems abstürzt oder anderweitig stehen bleibt. Auch müssen sie selbst während Software-Rollouts oder anderen Wartungsvorgängen weiterhin *verfügbar* sein. Und weil schließlich mehr und mehr Teile der Welt online gehen und solche Services nutzen, müssen diese gut *skalierbar* sein, damit ihre Kapazität mit der stetig wachsenden Nutzung mithalten kann, ohne dass das dahinterliegende verteilte System radikal umgeplant werden muss. In vielen Fällen geschieht dieses Wachsen (und Reduzieren) der Kapazität automatisch, sodass Ihre Anwendung möglichst effizient sein kann.

1. Brendan Burns et al., »Borg, Omega, and Kubernetes: Lessons from Three Container-Management Systems over a Decade«, *ACM Queue 14* (2016): 70–93, verfügbar unter *https://oreil.ly/ltE1B*.

Abhängig davon, wann und warum Sie dieses Buch in Ihren Händen halten, besitzen Sie vermutlich unterschiedlich viel Erfahrung mit Containern, verteilten Systemen und Kubernetes. Vielleicht planen Sie, Ihre Anwendung mithilfe der Infrastruktur einer öffentlichen Cloud zu bauen, in eigenen Data Centers oder in einer hybriden Umgebung. Aber unabhängig von Ihrer Erfahrung hoffen wir, dass Sie mit diesem Buch Kubernetes so gut wie möglich nutzen können.

Es gibt viele Gründe, warum die Leute Container und Container-APIs wie Kubernetes verwenden, aber wir sind der Meinung, dass sie alle auf einen dieser Vorteile zurückgeführt werden können:

- Schnelligkeit bei der Entwicklung
- Skalierbarkeit (sowohl der Software als auch der Teams)
- Abstrahieren Ihrer Infrastruktur
- Effizienz
- Cloud-natives Ökosystem

In den folgenden Abschnitten beschreiben wir, wie Kubernetes Ihnen dabei helfen kann, alle diese Features umzusetzen.

1.1 Schnelligkeit

Die Schnelligkeit ist bei so gut wie jeder aktuellen Software-Entwicklung von zentraler Bedeutung. Die Softwarebranche hat sich von verpackten CDs oder DVDs hin zu Software entwickelt, die als webbasierte Services über das Netz bereitgestellt wird, die stündlich Aktualisierungen erfahren. Diese sich verändernde Landschaft sorgt dafür, dass der Unterschied zwischen Ihnen und Ihrer Konkurrenz oft die Schnelligkeit ist, mit der Sie neue Komponenten und Features entwickeln und deployen können oder mit der es Ihnen möglich ist, auf von anderen entwickelte Innovationen zu reagieren.

Es sei aber darauf hingewiesen, dass Schnelligkeit nicht einfach die reine Geschwindigkeit ist. Während Ihre Anwender immer nach iterativen Verbesserungen Ausschau halten, sind sie trotzdem mehr an einem äußerst zuverlässigen Service interessiert. Früher war es in Ordnung, wenn ein Service jede Nacht um Mitternacht zu Wartungszwecken offline war. Aber heute erwarten alle Anwender durchgehende Uptime, auch wenn sich die Software, die das Ganze betreibt, fortlaufend ändert.

Konsequenterweise wird die Schnelligkeit nicht daran gemessen, wie viele Features Sie pro Stunde oder pro Tag liefern können, sondern wie viele Dinge Sie liefern können, während der Service weiterhin hochverfügbar ist.

Dafür können Container und Kubernetes die Werkzeuge liefern, die Sie benötigen, um sich schnell bewegen zu können, während Ihre Services verfügbar bleiben.

Die zentralen Konzepte, die das ermöglichen, sind:

- Immutabilität
- deklarative Konfiguration
- Online-Systeme, die sich selbst heilen
- Gemeinsam genutzte wiederverwendbare Bibliotheken und Tools

Diese Ideen arbeiten alle zusammen, um die Schnelligkeit, mit der Sie zuverlässig neue Software deployen können, radikal zu verbessern.

1.1.1 Der Wert der Immutabilität

Container und Kubernetes unterstützen Entwickler dabei, verteilte Systeme zu bauen, die sich an den Prinzipien der immutablen Infrastruktur orientieren. Bei einer solchen *immutablen* (unveränderlichen) Infrastruktur ändert sich ein Artefakt, sobald es einmal im System erzeugt wurde, nicht mehr durch die Anwender.

Klassisch wurden Computer- und Software-Systeme als *mutable* (änderbare) Infrastruktur betrachtet. Dabei werden Änderungen als inkrementelle Updates auf ein bestehendes System angewendet. Diese Updates können alle auf einmal vorgenommen werden oder auf einen langen Zeitraum verteilt sein. Ein System-Upgrade per `apt-get update` ist ein gutes Beispiel für ein Update eines mutablen Systems. Durch die Ausführung von `apt` werden nacheinander aktualisierte Binaries heruntergeladen, über die ältere Binaries kopiert und inkrementelle Anpassungen an Konfigurationsdateien vorgenommen werden. Bei einem mutablen System ist der aktuelle Status der Infrastruktur nicht als einzelnes Artefakt repräsentiert, sondern als Ansammlung inkrementeller Updates und Änderungen. Bei vielen Systemen kommen diese inkrementellen Updates nicht nur durch System-Updates zustande, sondern auch über Eingriffe durch den Nutzer. Zudem ist es in jedem von einem großen Team betreuten System sehr wahrscheinlich, dass diese Änderungen von vielen verschiedenen Leuten vorgenommen werden und sehr gerne nirgendwo dokumentiert wurden.

Im Gegensatz dazu wird bei einem immutablen System nicht eine Folge inkrementeller Updates und Änderungen angewendet, sondern es wird ein neues, vollständiges Image gebaut, und beim Update in einem einzigen Schritt das gesamte alte Image durch das neue ersetzt. Es gibt keine inkrementellen Änderungen. Wie Sie sich vorstellen können, ist das ein deutlicher Unterschied zum eher klassischen Wert des Konfigurations-Managements.

Um das in der Welt der Container konkreter zu machen, schauen Sie sich diese beiden verschiedenen Wege an, Ihre Software zu aktualisieren:

1. Sie können sich an einem Container anmelden, einen Befehl ausführen, um Ihre neue Software herunterzuladen, den alten Server abschießen und den neuen starten.
2. Sie können ein neues Container-Image bauen, es in eine Container-Registry schieben, den bestehenden Container abschießen und einen neuen starten.

Auf den ersten Blick mögen diese beiden Ansätze kaum unterscheidbar sein. Warum sorgt dann das Bauen eines neuen Containers für eine verbesserte Zuverlässigkeit?

Der entscheidende Unterschied ist das Artefakt, das Sie erstellen, und die Aufzeichnung, die beim Erstellen entsteht. Diese Aufzeichnung erleichtert es, die exakten Unterschiede in einer neuen Version zu verstehen. Geht etwas schief, können Sie herausfinden, was sich geändert hat und wie sich das korrigieren lässt.

Zudem sorgt das Bauen eines neuen Images statt des Anpassens eines bestehenden dafür, dass das alte Image immer noch verfügbar ist, sodass Sie dieses für ein Rollback nutzen können, wenn ein Fehler auftritt. Haben Sie im Gegensatz dazu Ihr neues Binary über ein bestehendes kopiert, ist solch ein Rollback nahezu unmöglich.

Immutable Container-Images bilden den Kern von allem, was Sie in Kubernetes bauen. Es ist möglich, im Notfall laufende Container anzupassen, aber das ist ein Antipattern, das nur in extremen Fällen eingesetzt werden sollte, wenn es keine anderen Optionen gibt (zum Beispiel, wenn es die einzige Möglichkeit ist, ein unternehmenskritisches Produktiv-System kurzfristig zu reparieren). Und selbst dann müssen die Änderungen später über ein deklaratives Konfigurations-Update dokumentiert werden, nachdem der Brand gelöscht wurde.

1.1.2 Deklarative Konfiguration

Immutabilität geht über die Container in Ihrem Cluster hinaus. Sie bezieht sich auch auf die Art und Weise, wie Sie Ihre Anwendung in Kubernetes beschreiben. Alles in Kubernetes ist ein *deklaratives Konfigurations-Objekt*, das den gewünschten Status des Systems repräsentiert. Es ist dann die Aufgabe von Kubernetes, sicherzustellen, dass der aktuelle Status der Wirklichkeit mit dem gewünschten Status übereinstimmt.

So wie bei mutabler und immutabler Infrastruktur ist die deklarative Konfiguration eine Alternative zur *imperativen* Konfiguration, bei der der Status der Welt durch das Ausführen einer Folge von Anweisungen beschrieben wird, statt den gewünschten Status zu deklarieren. Während imperative Befehle Aktionen definieren, definieren deklarative Konfigurationen einen Status.

Um diese beiden Ansätze zu verstehen, schauen Sie sich die Aufgabe an, drei Instanzen einer Software zu erzeugen. Bei einem imperativen Vorgehen würde die Konfiguration sagen: »Führe A aus, führe B aus, führe C aus.« Die entsprechende deklarative Konfiguration würde lauten: »Anzahl an Instanzen gleich drei.«

Da der Status der Welt beschrieben wird, muss eine deklarative Konfiguration nicht ausgeführt werden, um sie zu verstehen. Ihre Auswirkung ist konkret deklariert. Da die Auswirkungen einer deklarativen Konfiguration auch ohne Ausführen verstanden werden können, ist sie weniger fehleranfällig. Zudem können die klassischen Tools der Softwareentwicklung, wie Versionsverwaltung, Code-Review und Unit-Tests, bei deklarativen Konfigurationen auf eine Art und Weise eingesetzt werden, wie dies bei imperativen Anweisungen nie möglich wäre. Die Idee, deklarative Konfiguration unter Versionsverwaltung zu stellen, wird oft als »Infrastruktur als Code« bezeichnet.

In letzter Zeit hat die Idee von GitOps dazu geführt, dass die Praktiken von Infrastructure as Code mit einem Versionierungssystem als Source of Truth formalisiert wurden. Setzen Sie GitOps ein, geschehen Änderungen am Produktivumfeld komplett über Pushes in ein Git-Repository, die sich dann in Ihrem Cluster per Automation widerspiegeln. Tatsächlich wird Ihr produktives Kubernetes-Cluster letztendlich als Read-Only-Umgebung betrachtet. Zudem wird GitOps zunehmend in von der Cloud bereitgestellte Kubernetes-Services integriert, da Sie so am einfachsten Ihre Cloud-native Infrastruktur deklarativ managen können.

Die Kombination aus einem deklarativen Status in einem Versionierungssystem und den Fähigkeiten von Kubernetes, in der Realität diesen deklarativen Status zu erreichen, macht das Rollback einer Änderung ausgesprochen einfach. Es wird einfach der vorige deklarative Status des Systems gewählt. Bei imperativen Systemen ist das meist unmöglich, denn die imperativen Anweisungen beschreiben, wie Sie von Punkt A nach Punkt B gelangen, aber nur sehr selten sind die umgekehrten Anweisungen für den Rückweg enthalten.

1.1.3 Selbstheilende Systeme

Kubernetes ist ein selbstheilendes Online-System. Erhält es eine Konfiguration für einen gewünschten Status, nimmt es nicht nur einmalig die Schritte vor, um den aktuellen Status in den gewünschten Status zu überführen. Es achtet auch *kontinuierlich* darauf, dass der aktuelle Status und der gewünschte Status weiterhin übereinstimmen. Das heißt, Kubernetes initialisiert nicht nur Ihr System, sondern schützt es auch vor Fehlern oder Störungen, die es eventuell destabilisieren und die Zuverlässigkeit beeinträchtigen.

Bei einer klassischeren Reparatur durch Operatoren gibt es eine Reihe von manuellen Maßnahmen oder Eingriffe durch einen Menschen als Reaktion auf eine Warnung. Diese imperative Reparatur ist teurer (da dazu meist jemand Bereitschaftsdienst machen muss, um die Reparatur anzustoßen). Zudem ist sie im Allgemeinen langsamer, da ein Mensch oft erst aufwachen und sich anmelden muss, um reagieren zu können. Zudem ist sie weniger zuverlässig, da die imperative Folge von Reparationsschritten all die Probleme imperativen Managements mit sich bringt, die im vorigen Abschnitt beschrieben wurden. Selbstheilende Systeme wie Kubernetes reduzieren gleichzeitig die Last für die Operatoren und verbessern die Gesamtzuverlässigkeit des Systems, indem erprobte Reparaturen schneller durchgeführt werden.

Als Beispiel für dieses selbstheilende Verhalten nutzen wir wieder unseren gewünschten Status mit drei laufenden Instanzen in Kubernetes. Dabei legt es nicht nur diese Instanzen an, es stellt auch fortlaufend sicher, dass es immer genau drei Instanzen gibt. Erstellen Sie manuell eine vierte Instanz, wird Kubernetes eine zerstören, um die Anzahl wieder auf drei zu verringern. Beenden Sie manuell eine Instanz, erzeugt Kubernetes eine, um den gewünschten Status zu erreichen.

Selbstheilende Online-Systeme verbessern die Entwickler-Schnelligkeit, weil die Zeit und Energie, die Sie sonst für Operations und Wartung aufgewendet haben, nun für das Entwickeln und Testen neuer Features genutzt werden kann.

Für eine ausgefeiltere Form der Selbstheilung gab es jüngst deutliche Verbesserungen am *Operator*-Paradigma für Kubernetes. Dabei ist eine komplexere Logik zum Warten, Skalieren und Heilen einer bestimmten Software (zum Beispiel MySQL) in einer Operator-Anwendung verpackt, die als Container in einem Cluster läuft. Der Code im Operator ist dafür verantwortlich, den Status gezielter und ausgefeilter zu erkennen und Probleme zu beheben, als das mit den generischen Selbstheilungs-Fähigkeiten von Kubernetes möglich ist. Solche »Operatoren« werden später noch in Kapitel 17 behandelt.

1.2 Ihren Service und Ihre Teams skalieren

Wenn Ihr Produkt wächst, ist es unvermeidlich, dass Sie sowohl Ihre Software als auch das Team skalieren müssen, das diese entwickelt. Glücklicherweise kann Kubernetes dabei helfen, beide Ziele zu erreichen. Es nutzt dazu eine *entkoppelte* Architektur.

1.2.1 Entkoppeln

In einer entkoppelten Architektur ist jede Komponente von anderen Komponenten durch definierte APIs und Service-Load-Balancer getrennt. APIs und Load Balancer isolieren jedes Teil des Systems von den anderen. APIs dienen als Puffer zwischen dem Implementierer und dem Konsumenten und die Load Balancer liefern den Puffer zwischen den laufenden Instanzen jedes Service.

Das Entkoppeln von Komponenten über Load Balancer erleichtert das Skalieren des Programms, das hinter Ihrem Service steckt, weil das Erhöhen der Größe (und damit der Kapazität) des Programms erreicht werden kann, ohne dass eine der anderen Schichten Ihres Service angepasst oder umkonfiguriert werden muss.

Das Entkoppeln von Servern über APIs erleichtert das Skalieren des Entwicklungsteams, weil sich jedes Team auf einen einzelnen, kleineren *Microservice* mit einer verständlichen Oberfläche konzentrieren kann. Knackige APIs zwischen den Microservices beschränken die Menge an teamübergreifendem Kommunikationsoverhead, der notwendig ist, um Software zu bauen und zu deployen. Dieser Kommunikationsoverhead ist häufig der größte beschränkende Faktor, wenn man Teams skaliert.

1.2.2 Einfaches Skalieren für Anwendungen und Cluster

Wenn Sie ganz konkret Ihren Service skalieren müssen, sorgt die immutable, deklarative Natur von Kubernetes dafür, dass dieses Skalieren trivial zu implementieren ist. Weil Ihre Container immutabel sind und die Anzahl der Instanzen einfach eine Zahl in einer deklarativen Konfiguration ist, geht es beim Hochskalieren Ihres Service schlicht darum, eine Zahl in einer Konfigurationsdatei zu ändern, diesen neuen deklarativen Status Kubernetes mitzuteilen und es sich dann um den Rest kümmern zu lassen. Alternativ können Sie auch ein Autoscaling einrichten und das Ganze von Kubernetes erledigen lassen.

Natürlich wird bei dieser Art von Skalierung davon ausgegangen, dass es in Ihrem Cluster Ressourcen gibt, die genutzt werden können. Manchmal müssen Sie aber auch das Cluster selbst skalieren. Auch hier hilft Kubernetes. Weil viele Maschinen in einem Cluster identisch mit anderen sind und die Anwendung selbst von den Maschinendetails durch die Container entkoppelt ist, geht es beim Hinzufügen von Ressourcen zum Cluster nur darum, eine neue Maschine der gleichen Klasse mit einem Image zu versehen und sie in das Cluster einzuhängen. Das lässt sich über ein paar einfache Befehle oder ein vorgefertigtes Image erreichen.

Eine der Herausforderungen beim Skalieren von Rechner-Ressourcen ist die Vorhersage der Verwendung. Lassen Sie Ihr Cluster auf einer Infrastruktur aus echten Rechnern laufen, wird die Zeit zum Bereitstellen einer neuen Maschine in Tagen oder Wochen gemessen. Sowohl bei einer »echten« wie auch bei einer Cloud-Infrastruktur ist die Vorhersage zukünftiger Kosten schwierig, weil es schwerfällt, das Wachstum und die Skalierungsanforderungen bestimmter Anwendungen zu prognostizieren.

Kubernetes kann das Vorhersagen zukünftiger Kosten vereinfachen. Um das zu verstehen, stellen Sie sich das Vergrößern von den drei Teams A, B und C vor. Aus Erfahrung wissen Sie, dass das Wachstum jedes Teams sehr variabel ist und sich daher schlecht vorhersagen lässt. Provisionieren Sie für jeden Service individuelle Maschinen, haben Sie keine andere Wahl, als Ihre Vorhersagen auf der maximal zu erwartenden Wachstumsrate für jeden Service basieren zu lassen, da die Rechner für das eine Team nicht für ein anderes Team eingesetzt werden können. Nutzen Sie stattdessen Kubernetes, um die Teams von den spezifischen Rechnern zu entkoppeln, können Sie das Wachstum basierend auf dem Gesamtwachstum aller drei Services vorhersagen. Durch das Kombinieren von drei variablen Wachstumsraten zu einer einzigen Rate reduzieren Sie statistisches Rauschen und sorgen für eine zuverlässigere Vorhersage des zu erwartenden Wachstums. Zudem bedeutet das Entkoppeln spezifischer Maschinen von den Teams, dass diese auch Anteile der Rechner mit anderen teilen können und damit den Overhead noch weiter verringern, der mit dem Vorhersagen der Wachstumsanforderungen von Rechenressourcen verbunden ist.

Schließlich ermöglicht es Kubernetes, Ressourcen automatisch (nach oben und unten) skalieren zu können. Insbesondere in einer Cloud-Umgebung, in der sich neue Maschinen per API erstellen lassen, sorgt eine Kombination von Kubernetes mit einem Autoscaling für die Anwendungen und die Cluster selbst dafür, dass Sie immer nur die Kosten für die aktuelle Last zu tragen haben.

1.2.3 Entwicklungs-Teams mit Microservices skalieren

Wie sich in einer Reihe von Untersuchungen gezeigt hat, ist die ideale Teamgröße das »Zwei-Pizza-Team« – etwa sechs bis acht Personen –, weil diese Gruppengröße häufig zu einer guten Wissensverteilung, schnellen Entscheidungen und einem teamweiten Verständnis für die Aufgaben führt. Größere Teams tendieren dazu, mit Hierarchien, schlechter Sichtbarkeit und Machtkämpfen zu hadern, was ihre Agilität und ihren Erfolg einschränkt.

Für viele Projekte sind aber deutlich mehr Ressourcen notwendig, um erfolgreich zu sein und die Ziele zu erreichen. Daher gibt es einen Konflikt zwischen der idealen Teamgröße für gute Agilität und der notwendigen Teamgröße für das Erreichen der Produktziele.

Die übliche Lösung für diese Spannung ist das Entwickeln in entkoppelten, serviceorientierten Teams, die jeweils einen einzelnen Microservice bauen. Jedes kleine Team ist für das Design und die Auslieferung eines Service verantwortlich, der von anderen kleinen Teams konsumiert wird. Das Zusammenführen all dieser Services bildet dann schließlich die Implementierung der Schnittstelle des Gesamtprodukts.

Kubernetes stellt eine Reihe von Abstraktionen und APIs bereit, die es leicht machen, diese entkoppelte Microservice-Architektur zu bauen.

- *Pods* – Gruppen von Containern – können von verschiedenen Teams entwickelte Container-Images zu einer einzelnen deploybaren Einheit verbinden.
- Kubernetes-*Services* bieten Load Balancing, Naming und Discovery, um einen Microservice von anderen zu isolieren.
- *Namensräume* bieten Isolation und Zugriffskontrolle, sodass jeder Microservice steuern kann, inwieweit andere Services mit ihm interagieren können.
- *Ingress*-Objekte dienen als einfach zu nutzendes Frontend, das mehrere Microservices zu einer einzelnen externalisierten API kombinieren kann.

Und schließlich sorgt das Entkoppeln des Anwendungs-Container-Image und der Maschine dafür, dass verschiedene Microservices auf derselben Maschine laufen können, ohne sich gegenseitig ins Gehege zu kommen, was den Overhead und die Kosten der Microservice-Architektur verringert. Die Healthchecking- und Rollout-Features von Kubernetes garantieren ein konsistentes Vorgehen beim Anwendungs-Rollout und eine Zuverlässigkeit, die sicherstellt, dass eine Zunahme von Microservice-Teams nicht auch zu einer Zunahme unterschiedlicher Vorgehensweisen beim Lebenszyklus für die Service-Erstellung und in Operations führt.

1.2.4 Konsistenz und Skalierung durch Separation of Concerns

Neben der Konsistenz, die Kubernetes in Operations mitbringt, sorgt das Entkoppeln und die Separation of Concerns, die durch den Kubernetes-Stack entsteht, für eine deutlich höhere Konsistenz bei den unteren Schichten Ihrer Infrastruktur. So kann Operations viele Maschinen mit einem kleinen, fokussierten Team verwalten. Sie haben schon viel über das Entkoppeln von Anwendungs-Containern und der Maschine beziehungsweise dem Betriebssystem (OS) gelesen, aber ein wichtiger Aspekt dieser Trennung ist, dass die Orchestrierungs-API für die Container zu einem klaren Vertrag wird, der die Verantwortlichkeiten von Anwendungs-Operator und Cluster-Orchestrierungs-Operator aufteilt. Wir nennen dies die »Not my monkey, not my circus«-Linie. Der Anwendungs-Entwickler verlässt sich auf das Service-Level Agreement (SLA) der Container-Orchestrierungs-API, ohne sich um die Details zu